(the arrows labeled three ($_3$) and four ($_4$)). Via the tables in the activation context **302**, the runtime version-matching mechanism **500** will either return (e.g., as a return parameter, with the arrow labeled five (5)) the path and filename of the version-specific assembly, or a not-found status or the like (and/or the path and filename of the default file) to the activation API that called it, in which case the activation API will load the default assembly. As represented in **FIG. 5** by the arrows labeled six (6) through eight (8), the activation API **300**$_1$ accesses the assembly cache **212** to load the correct version of the assembly. Note that alternatively, the runtime version-matching mechanism **500** may access the assembly cache **212** on behalf of the calling API **300**$_1$ to obtain and/or load the correct version, which is either the one listed in the records or the default version if none was found. Further, note that privatized assemblies may be loaded from the application directory. In any event, the correct assembly version **510** is loaded and the activation API **300**$_1$ returns from the call to the application **200**, (the arrow labeled nine (9)), and the application can use the loaded assembly **510** (the arrow labeled ten (10)).

[**0083**] Turning to an explanation of the operation of the present invention with particular reference to the flow diagrams of FIGS. **6-7** and **9**, as described above, the mapping of an application to a specified version of an assembly or other data structures essentially comprises two phases, an initialization phase in the first alternative mode (FIGS. **6-7**) and a runtime phase (**FIG. 9**). In general, the initialization phase constructs the activation context, if needed, that maps version-independent assemblies to version-specific assemblies based on the dependencies and other instructions provided in the various manifests and configurations. Then, during a runtime phase, (**FIG. 9**), the activation context is accessed as needed to rapidly locate and load the appropriate versions when an application requests an assembly to which it needs to be bound.

[**0084**] FIGS. **6-7** represent example steps that may be taken in the first alternative mode during the initialization (pre-application execution) phase to construct the activation context, if needed. When creating a new process, the binding/initialization mechanism **304** (e.g., of the operating system) checks for an application manifest in same file system directory as the calling executable, as represented in **FIG. 6** by step **600**. If an application manifest does not exist, the binding/initialization mechanism **304** handles its absence in another manner, (step **602**), e.g., the operating system may search for it, download it, and/or essentially give the application default versions during runtime, such as by first loading any requested assembly from the application's own directory when one is present, and otherwise using the default assemblies from the assembly cache.

[**0085**] When step **600** determines that an application manifest exists, the binding/initialization mechanism **304** (**FIG. 3**) preferably branches to step **610** to create the activation tables. Alternatively, if activation contexts may be preserved rather than recomputed each runtime, the binding/initialization mechanism **304** may check for an existing activation context (e.g., **302**) for the application. If an existing activation context is found, step **604** branches to step **606** to validate it, otherwise step **604** branches to step **610**. Step **606** checks the activation context to determine if it is coherent with current configuration, and if so, the existing activation context **302** can be used (step **608**) and the initialization process ends. If alternatively the activation context **302** is not coherent with current configuration at step **606**, for example, because a more recent configuration has been provided to the system, the initialization process continues to step **610** to recompute a new activation context **302**.

[**0086**] In the event that the initialization process continues to step **610** to create the activation context, step **610** represents obtaining the binding information from the application manifest. Steps **610**, **612**, **614** and **616** of **FIG. 6** are executed, along with the steps of **FIG. 7**, essentially to walk through the application manifest, configurations and any assembly manifests in order to build up a dependency graph, including replacing assembly information (e.g., maintained as nodes in the graph) according to configurations in the dependency graph and adding any new nodes to include the dependencies of any assembly manifests.

[**0087**] By way of example, **FIG. 7** operates once the application manifest has it dependent assemblies added to the dependency graph (step **610** of **FIG. 6**) and a requested assembly version (node) therein has been selected (step **612**, e.g., via a top-down, left-to-right or other suitable progression) for processing.

[**0088**] At step **700**, a test is performed to determine whether the assembly has a publisher configuration associated therewith, e.g., in the global assembly cache. If not, step **700** branches ahead to test for an application configuration at step **706**, described below. If a publisher configuration is found at step **700**, step **700** branches to step **702** wherein the publisher configuration is interpreted to determine whether there is an instruction therein for replacing the assembly version that is currently under evaluation, i.e., the one currently selected in the dependency graph either as originally specified in the manifest. If a replacement instruction is found, step **702** branches to step **704** wherein a dependency graph is altered to reflect the replacement, otherwise step **702** effectively bypasses step **704**. By way of example, **FIG. 8** shows a dependency graph **800** in which a node representing an assembly, such as the node **N1**, has been replaced by a node **N2**. Note that a list or other data structure may be used instead of a dependency graph.

[**0089**] At step **706**, a test is performed to determine whether the application has an application configuration associated therewith, e.g., in the application directory. If an application configuration is found at step **706**, step **706** branches to step **708** wherein the application configuration is interpreted to determine whether there is an instruction therein for replacing the assembly version that is currently under evaluation, i.e., currently selected in the dependency graph by identification in the manifest or as overridden by the publisher policy. If such a relevant replacement instruction is found, step **708** branches to step **710** wherein the dependency graph is altered to reflect the replacement.

[**0090**] At this time, the appropriate assembly version is known, as specified in the manifest and as altered via any configuration instructions, as described above. Step **712** enumerates any dependencies in the assembly manifest that corresponds to this appropriate assembly, e.g., to add dependent nodes to the dependency graph. Note that if an assembly representation (node) is already in the graph for a given assembly, a pointer from the node may be added to show the dependency rather than place a new node in the dependency